Express Mail No. EL844351271US

PATENT APPLICATION OF

SERGEJ B. GASHKOV
Rossoshanskaja Ul.,D. 7K. 1A KV.82
Moscow, 113535 Russia
Citizenship: RUSSIA

ALEXANDER E. ANDREEV
2774 Glen Firth Drive, San Jose, CA 95133
Citizenship: RUSSIA

AIGUO LU
7920 McClellan Road #4, Cupertino, CA 95014
Citizenship: CHINA

ENTITLED

OPTIMIZATION OF ADDER BASED CIRCUIT ARCHITECTURE

Docket No. L13.12-0173/00-630

OPTIMIZATION OF AN ADDER BASED CIRCUIT ARCHITECTURE

CROSS REFERENCE TO RELATED APPLICATION

This application is related to Application No. (L13.12-0172/00-629) filed on even date herewith for "Optimization of Comparator Architecture" by Sergej B. Gashkov, Alexander E. Andreev and Aiguo Lu and assigned to the same assignee as the present invention, which application is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to digital adders, and particularly to adders for use in application specific integrated circuits (ASICs).

BACKGROUND OF THE INVENTION

Adders are widely used in ASICs and represent one of the most common and basic circuit functions of general purpose digital computers and processors. Adder based circuits include adders, subtractors, adder-subtractors, incrementors, decrementors, incrementor-decrementors, and absolute value calculators, to name a few. Nearly every datapath module of nearly every digital IC includes adder based circuits. Thus, adders are crucial to the operation of computers, processors and controllers.

Each element on a chip, be it an element of an adder or some other device, is derived from a library of cells, the library being technology dependent based on the processing technology used to fabricate the IC chip. Each cell requires some space (area) on

the chip, and the cells forming the element require some depth to the chip. Consequently, the elements formed by the cells require some minimum amount of volume on the chip.

Most adders are implemented to perform a Boolean function such that S_n = A_n+B_n, where S_n is the digital output and A_n and B_n are the digital inputs. Most adders are composed of AND, OR and Exclusive-NOR (XNOR) elements. However, these elements often require considerable space and depth, and signal propagation through the element may cause timing delays.

As integrated circuit processing continues to advance, the need increases for smaller, faster adders. The present invention is directed to this need.

The aforementioned application of Gashkov et al. describes a comparator architecture for ICs based on a Fibonacci series. The resulting circuit is smaller and has less depth, and hence less delay, than a corresponding comparator of traditional design. The present invention extends that concept to adders for ICs.

SUMMARY OF THE INVENTION

In accordance with the present invention, an adder is implemented as three modules, an input module, a carry module, and an output module. The input and/or output modules are optimized based on a global analysis of the Boolean representations of

functions to optimize circuit area and depth and reduce delay. The carry module is based on a Fibonacci series.

In one embodiment, an adder based circuit is embodied in an integrated circuit. The adder based circuit comprises an input module that receives inputs A[i] and B[i] to generate U[i] = A[i]&B[i] and either V[i] = A[i]∨B[i] or V[i] = A[i]⊕B[i]. A carry module is responsive to U[i] and V[i] to generate carry functions. An output module is responsive to the U[i], V[i] and carry functions to provide an output function $S=\sum_{i=0}^{n-1}2^{i}S[i] = \sum_{i=0}^{n-1}2^{i}A[i] + \sum_{i-0}^{n-1}2^{i}B[i]$. The carry module has a minimal depth defined by a recursive expansion of at least one carry function associated with the carry module based at least in part on a variable k, where k = F_l and n-k = F_{l-1} and where l satisfies F_l < n = F_{l+1}, {F_l} is a Fibonacci series and n is the number of bits of at least one of U[i] and V[i].

In various embodiments, the adder based circuit functions as a subtractor, adder-subtractor, incrementor, decrementor, increment-decrementor and absolute value calculator, depending on elements applied to the inputs and outputs.

In some embodiments, an adder is designed for an integrated circuit. At least one output function of a carry module of the adder is defined in terms of a Fibonacci series. The output functions are

recursively expanded to find a minimum parameter of the Fibonacci series.

In one embodiment, recursive functions are defined as h'_l = h_l(U[k+1],U[k+2],V[k+2],..., U[k+1],V[k+1]) and v'_l = V[k+1]&,...,&V[k+1], based on the carry functions, where k=F_l and n-k=F_{l-1}, l satisfies F_l < n = F_{l+1}, {F_l} is the Fibonacci series defined recursively from the equality F_{l+1}=F{l}+F_{l+1} and n is the number of bits of an input to the carry module. The recursive functions are recursively expanded to minimize l.

In some embodiments, negations are optimally distributed in the carry module by identifying a set of delay vectors, recursively comparing the delay vectors to derive a set of minimum vectors, and selecting a vector with a minimum norm from the set of minimum vectors.

In other embodiments the depth of the adder is minimized by recursively expanding expressions

DH_n = min{max(DH_{n-k}+1,Dh_k+2,DH_k}} and

Dh_n = min{max(Dh_{n-k}+1,Dh_k+2}},

where DH_k and Dh_k are based on vectors of depths U[i] for $0 \le i \le k-1$ and DH_{n-k} and Dh_{n-k} are based on vectors for $k \le i \le n-1$. A value of k is selected based on a minimum of at least one of DH_n and Dh_n.

In other embodiments, the fanout depth of the adder based circuit is minimized by defining recursive functions

$$H^i_k = h^{i+l}_{k-l} \vee v^{i+l}_{k-l} \& h^i_l$$

and

$$v^i_l = v^{i+l}_{k^l} \& v^i_l$$

based on the output function,

where $k=F_l$ and $n-k=F_{l-1}$, l satisfies $F_l<n=F_{l+1}$, $\{F_l\}$ is the Fibonacci series defined from the equality $F_{l+1}=F\{l\}+F_{l+1}$. The recursive functions are recursively expanded to minimize l.

In other embodiments, the invention is manifest in a computer readable program containing code that, when executed by a computer, causes the computer to perform the process steps to design an adder based circuit for an integrated circuit based on a Fibonacci series.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a functional diagram of a basic adder circuit according to one embodiment of the present invention.

FIGS. 2A and 2B are functional diagrams of the input and output modules of the circuit shown in FIG. 1.

FIG. 3 is a functional diagram of a carry module according to one embodiment of the present invention for use in the circuit shown in FIG. 1.

FIG. 4 is a functional diagram of a subcircuit used in the carry module shown in FIG. 3.

FIG. 5 is a functional diagram of a carry module according to another embodiment of the present invention for use in the circuit shown in FIG. 1.

FIG. 6 is a functional diagram of one embodiment of a subcircuit used in the carry module shown in FIG. 5.

FIGS. 7A and 7B are functional diagrams of circuits used in the subcircuit of FIG. 8.

FIG. 8 is a functional diagram of another embodiment of a subcircuit used in the carry module shown in FIG. 5.

FIGS. 9 and 10 are functional diagrams of other embodiments of carry modules used in the adder based circuit illustrated in FIG. 1

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following, Section 1 describes Boolean representations of seven different adder-based circuits are described to demonstrate how they may be implemented from a common adder circuit by modifying the input and/or output modules of an adder according to the present invention. In Section 2, implementation of an adder according to the present invention is described, and Section 3 describes optimization techniques for the adder.

1. BOOLEAN REPRESENTATIONS OF ARITHMETIC CIRCUITS

1.1. Adder

An adder is a circuit with inputs A[0],B[0],...,A[n-1],B[n-1] and outputs

S[0],...,S[n], such that the sum of numbers $A=\sum_{i=0}^{n-1}2^{i}A[i]$ and $B=\sum_{i=0}^{n-1}2^{i}B[i]$ is equal to $S=\sum_{i=0}^{n}2^{i}S[i]$. In accordance with the present invention, an adder is implemented as three modules: an input module, a carry module, and an output module. The input and/or output modules are optimized based on a global analysis of the Boolean representations of functions to optimize circuit area and depth and reduce delay. The carry module is based on a Fibonacci series.

The input module receives inputs A[i] and B[i] to generate U[i] = A[i]&B[i] and either V[i] = A[i]∨B[i] or V[i] = A[i]⊕B[i]. The carry module is responsive to U[i] and V[i] to generate a carry functions W[i]. The output module is responsive to the U[i], V[i] and W[i] to provide an output function S[i].

Considering first a special class of subtractor circuit that subtracts an n-bit number B having bits (B[n-1],...,B[O]) from an (n+1)-bit number A having bit A[n]=1 and bits (A[n-1],..., A[O]), so A > B, the Boolean representation is:

$$\sum_{i=0}^{n-1}2^{i}A[i]+2^{n}-\sum_{i=0}^{n-1}2^{i}B[i]=2^{n+1}-1-\sum_{i=0}^{n-1}2^{i}\neg A[i]-\sum_{i=0}^{n-1}2^{i}B[i]$$

$$=2^{n+1}-1-\sum_{i=0}^{n}2^{i}S[i]$$

$$=\sum_{i=0}^{n}2^{i}\neg S[i],$$

where symbol ¬ denotes logical negation. This special circuit can be obtained from an adder circuit by connecting inverters to inputs A[O],..., A[n-1] of the adder (but not to inputs B[O],..., B[n-1]) and by connecting inverters to outputs S[O],..., S[n]. The last output, ¬S[n], is equal to 1 if and only if number A, having digits A[O],..., A[n], is greater than or equal to number B, having digits B[0],..., B[n-1], that is, A ≥ B.

In the aforementioned Gashkov et al. application, we describe a comparator that provides two outputs LT_LE and GE_GT. If the number of bits in A is greater than or equal to the number of bits in B, GE_GT = 1. Therefore, if GE_GT = 1, A ≥ B. In an adder, adding digit-by-digit numbers A and B generate a sequence of carries W[1],...,W[n+1], which is recursively calculated as

$$W[i+1] = maj(A[i],B[i],W[i]) = U[i] \vee V[i] \& W[i],$$

where maj(A,B,W) is a majority function, namely

$$maj(A,B,W) = A\&B \vee A\&W \vee B\&W = U \vee V\&W = U \oplus P\&W,$$

where U=A&B, V=A∨B, P=A⊕B, W[0]=0, W[1]=U[1], and S[i] = A[i]⊕B[i]⊕W[i] = P[i]⊕W[i], P[i] = A[i]⊕B[i], for 1 ≤ i ≤ n and S[n] = W[n]. (The function A⊕B can also be represented by XOR(A,B). The symbol & denotes logical conjunction and the symbol ∨ denotes logical disjunction.) By induction,

$$W[n](A[O],B[O],...,A[n-1],B[n-1]) = h\_n(U[O],U[1],V[1],...,U[n-1],V[n-1]),$$

U[i] = A[i]&B[i],

V[i] = A[i]∨B[i], 0 ≤ i ≤ n-1, where the function h_n is determined from the equality

h_n(U[0],U[1],V[i],...,U[n-1],V[n-1])

= U[n-1]∨V[n-1]&U[n-2]∨V[n-1]&V[n-2]&U[n-3]

∨...∨V[n-1]&V[n-2]...V[i]&U[0].

(It will be appreciated that the above equality is the same as in the case of the comparator described in the aforementioned Gashkov et al. application.) Usually V[i] will be realized by the function A[i]⊕B[i].

The carries are calculated as:

W[i+1] = W[i+1](A[0],B[0],...,A[i],B[i])

= h_{i+1}(U[0],U[1],V[1],...,U[i],V[i]),

1 ≤ i ≤ n-1, and

W[1](A[0],B[0]) = h_{i}(U[0])

= U[0]

= A[0]&B[0].

The standard adder will have an additional input CI (input carry) and an additional output CO (output carry). The remaining outputs will realize all digits of the sum $\sum_{i=0}^{n-1} 2^i A[i] + \sum_{i=0}^{n-1} 2^i B[i] + CI$. To include the input carry signal into the adder, it is necessary to change the equality from S[0] = A[0]⊕B[0] to S[0] = A[0]⊕B[0]⊕CI and from W[1] = A[0]&B[0] to

W[1] = maj(A[0],B[0],CI)

= U[0]∨V[0]&CI

= h_2(CI,U[0],V[0])

and from

W[i+1] = h_{i+1}(U[0],U[1],V[1],...,U[i],V[i]),

where $1 \leq i \leq n-1$

to

W[i+1] = h_{i+2}(CI,U[0],V[0],...,U[i],V[i]),

where i = 1,...,n-1.

In particular, the carry at output CO will be

W[n] = h_{n+1}(CI,U[O],V[O],...,U[n-1],V[n-1]).

To reduce circuit depth, and hence signal delays, it is preferred to employ an element composed of an OR element and inverter in place of XNOR elements: XNOR(a,b) = OR(NOT(a),b)).

1.2. Subtractor

A subtractor circuit is implemented based on the following identities:

$$\sum_{i=0}^{n-1} 2^i A[i] - \sum_{i=0}^{n-1} 2^i B[i] - CI = 2^n - 1 - \sum_{i=0}^{n-1} 2^i \neg A[i] - \sum_{i=0}^{n-1} 2^i B[i] - CI$$

$$= 2^{n+1} - 1 - \sum_{i=0}^{n} 2^i S[i] - 2^n$$

$$= \sum_{i=0}^{n} 2^i \neg S[i] - 2^n$$

which $= \sum_{i=0}^{n-1} 2^i \neg S[i] - 2^n$, if S[n]=1

and $= \sum_{i=0}^{n-1} 2^i \neg S[i]$, if S[n]=0.

The subtractor can be obtained from an adder by attaching inverters to inputs A[0],...,A[n-1] (but not to inputs B[0],...,B[n-1],CI) and by attaching

inverters to outputs S[0],...,S[n]. Topologically the circuit can be kept the same if XOR elements on outputs of the circuit are replaced with XNOR instead of adding the inverters. As mentioned above, however, it is preferred to employ OR(NOT(a),b) in place of the XNOR element.

1.3. Adder-Subtractor

An adder-subtractor circuit is one having an additional input ADD_SUB. If ADD_SUB = 0, the circuit operates as an adder; if ADD_SUB = 1, the circuit operates as a subtractor. To construct the circuit inputs A[i], i = 0,...,n-1 and all outputs of the adder are connected through additional XOR elements, with each additional XOR element having a common input ADD_SUB. If ADD_SUB = 0, the circuit will operate in the adder function because the additional XOR elements will simply pass the state of the input A[i] and output S[i] bits. If ADD_SUB = 1, the circuit will operate as a subtractor because the additional XOR elements will operate as inverters. The complexity of the circuit will increase by 2n + 1 and the depth will increase by 2 when compared with an adder.

1.4. Incrementor

An incrementor with n inputs A[0],...,A[n-1] can be represented as $\sum_{i=0}^{n} 2^i S[i] = \sum_{i=0}^{n-1} 2^i A[i] + 1$. Therefore an incrementor is a simplification of an adder, with the second operand is equal to constant 1. If numbers A

and B are added in sequence a sequence of carries W[0],...,W[n], is recursively calculated based on

$W[i] = U[i] \vee V[i]W[i-1]$, $i > 0$, $W[0] = U[0]$ where

$U[i] = A[i]\&B[i]$, $W[i] = A[i] \vee B[i]$ and the output functions are

$S[i] = A[i] \oplus B[i] \oplus W[i-1]$, $1 \leq i \leq n$, and

$S[0] = A[0] \oplus B[0]$, $S[n] = W[n]$.

The sequence of the carries is calculated as $W[0] = A[0]$, $W[i] = A[i]$, $W[i-1] = A[i],...,A[0]$, for $i = 1,...,n-1$. The outputs represent the functions $S[0] = \neg A[0]$, $S[i] = A[i] \oplus W[i-1]$, for $i = 1,...,n-1$, and $S[n] = W[n-1]$, because vector (B[0],..., B[n-1]) = (1,0,...,0). Therefore the incrementor can be implemented by using the input module of the adder providing the conjunctions $W[i] = A[i]\&...\&A[0])$ and the output module of the adder. The carry module is modified by simply coupling the output of the input module to the XOR elements of the output module.

It is possible to reduce the delay of this incrementor circuit by using negation instead of conjunctions for W[i], using XOR elements at the outputs instead of XNOR elements, and forming the input circuits of OR elements with outputs coupled to NAND elements of the adder. One technique for distributing negations is described in the aforementioned Gashkov et al. application.

It is possible to further reduce delay by duplicate elements of the circuit so that elements performing negation functions are adjacent the

corresponding elements and the circuit will be constructed only using elements NAND and NOR. One of the two mutually-inverse duplicate elements (having minimum delay) is selected and elements that are useless can be deleted from the circuit. The delay can be further reduced if the multiple input elements NAND and NOR are used.

1.5. Decrementor

A decrementor is similar to the subtractor with the second operand equal to a constant 1. To implement a decrementor, a subtractor is arranged to subtract from the number $A=\sum_{i=0}^{n-1}2^{i}A[i]$ by negation on the inputs and output, as if $\sum_{i=0}^{n}2^{i}S[i]=\sum_{i=0}^{n-1}2^{i}A[i]+1$ that

$$\sum_{i=0}^{n}2^{i}\neg S[i]=2^{n+1}-1-\sum_{i=0}^{n}2^{i}S[i]=2^{n+1}-1-\sum_{i=0}^{n-1}2^{i}A[i]-1$$

$$=2^{n}+\sum_{i=0}^{n-1}2^{i}\neg A[i]-1\,.$$

The decrementor is implemented similar to the incrementor, and it is not necessary to connect inverters with inputs as is necessary for the subtractor. Nor is it necessary to connect inverters to the output of circuit. Instead, the decrementor is formed by replacing the XNOR elements of the incrementor with XOR elements. Hence, decrementor and incrementor circuits have identical topology and can be composed from the dual elements in the

aforementioned Gashkov et al. application to realize the dual functions.

1.6. Incrementor-Decrementor

The incrementor-decrementor is a circuit having an additional input s. If s = 0 the circuit operates as an incrementor; if s = 1 the circuit operates as a decrementor. To construct this circuit, the inputs and outputs of incrementor are connected through additional XOR elements, each having a common input s. If s = 0, the circuit will operate as an incrementor because the additional XOR elements will simply pass the state of the input A[i] or S[i] bits, respectively. If s = 1 the circuit will operate as a decrementor because the additional XOR elements will operate as inverters. The complexity of the incrementor-decrementor circuit will increase by 2n and the depth will increase by 2, as compared to an incrementor.

1.7. Absolute Value Calculator

By convention, the sign bit of a signed number is the most significant bit (n-1). If the n-1 bit is a "1", the number is considered negative; if the n-1 bit is a "0", the number is considered nonnegative. For an absolute value calculator, if the sign bit A[n-1] = 0, the circuit simply passes inputs to outputs without any changes. If the sign bit A[n-1] = 1, the circuit calculates the digits of number as $2^n - \sum_{i=0}^{n-1} 2^i A[i] = 1 + \sum_{i=0}^{n-2} 2^i \neg A[i]$. Therefore, the absolute value

calculator operates as an incrementor with inverters on the n-1 inputs. The circuit is topologically the same as the incrementor. The outputs are connected to multiplexers controlled by the input A[n-1] so if A[n-1] = 0, the multiplexer outputs are passed without change. If A[n-1] = 1, the multiplexer causes the outputs of the incrementor to be passed. The complexity of the absolute value calculator circuit is increased on the n-1 path, and depth increases by 1 over the incrementor.

2. IMPLEMENTATION OF AN ADDER

FIG. 1 illustrates an adder circuit consisting of three modules: input module 10, carry module 12 and output module 14. Input module 10 performs the functions

$$U[i] = A[i] \& B[i] \text{ and}$$

$$V[i] = A[i] \vee B[i], \text{ for } 0 \leq i \leq n-1.$$

Instead of $A[i] \vee B[i]$, it is also possible to use $A[i] \oplus B[i]$, in which case U[i] and V[i] can be simultaneously realized by a half-adder element. One embodiment of input module 10 is illustrated in greater detail in FIG. 2A as including AND elements 16 and OR elements 18.

Output module 14 provides the output functions of the adder of

$$S[i] = A[i] \oplus B[i] \oplus W[i]$$

$$= P[i] \oplus W[i], \text{ where } P[i] = A[i] \oplus B[i],$$

for $1 \leq i \leq n$, $S[n] = W[n]$. In one embodiment shown in FIG. 2B, the output circuit consists of 3-input XOR

elements EO3(a,b,c) = a⊕b⊕c. It is possible to apply 2-input XOR elements EO(a,b) = a⊕b = XOR(a,b) for optimization if a half-adder is used for input module 10. In such a case P[i] = A[i]⊕B[i] has already been implemented at the input module, and the output module comprises EO(a,b) elements to perform the function S[i] = P[i]⊕W[i].

The principal module of the adder is carry module 12 which realizes functions h_1,...,h_{n+1}. For sake of brevity n will be used instead of n+1 and carry module 12 will be denoted as h_n. Recursive generation of the carry module H_n = {h_1,...,h_n} is based on application of identities h_{k+l} = h'_l∨v'_l&h_k = AO21(v'_l,h_k,h'_l), where h'_l = h_l(U[k+1],U[k+2],V[k+2],...,U[k+l],V[k+l]), and v'_{l} = v_{l,k} = V[k+l]&...&V[k+1] for l = 1,...,n-k. As shown in FIG. 3, in one form module H_n consists of modules H_k = {h_1,..., h_k} and HV_{n-k} = {h'_1,v'_1,...,h'_{n-k},v'_{n-k,}} and also their connecting modules AO21 that performs the function A021(a,b,c) = (a&b)∨c, shown in FIG. 4.

A carry module, shown in FIG. 5 and designated HV_n, derives carries simultaneously with conjunctions {h_1,v_1,...,h_n,v_n}, where v_k = v_{k,n} = V[n]&...&V[n-k+1]. Carry module HV_n is recursively generated from modules HV_k = h_1,v_1,...,h_k,v_k and HV_{n-k} = h')1,v'_1,...,h'_{n-k},v'_{n-k} and their connecting module C_n,k. Module C_n,k is shown in greater

detail in FIG. 6 consisting of elements AND and AO21 (FIG. 4), due to identities

$$H_{k+l} = h'_l \vee v'_ \& h_k = AO21(v'_l, h_k, h'_l),$$

$$h'l = h_l(U\{k+1\}, U[k+2, V[k+2], \ldots, U\{k+l\}, V[k+l],$$

$$v'_l = v_{l, k} = V[k+l] \& \ldots \& V[k+1],$$

$$\text{for } l = 1, \ldots, n-k$$

$$V_{k+l} = v_{l,k} \& v_k$$

$$= (V[k+l] \& \ldots \& V[k+1]) \& (V[k] \& \ldots \& V[1])$$

$$= V[k+l] \& \ldots \& V[1].$$

The optimal choice of parameter k on each step of the recursion for the purpose of minimizing depths of a circuit is the same as for a comparator described in the aforementioned Gashkov et al. application. More particularly, k is selected from the closed interval between F_i to $n-F_{i-1} = F_i$, where F_i is a number selected from a Fibonacci series. Most particularly, the circuit depth is minimized by selecting k so that $k \geq F_l$, $n-k \leq F_{l-1}$, where the number l satisfies $F_l < n \leq F_{l+1}$, and the Fibonacci series F_l is defined recursively from the equality $F_{l+1} = F_l + F_{l-1}$ using initial values $F_1 = 1$, $F_0 = 0$. The value of k is determined uniquely when n is equal to a Fibonacci number; in other cases k is selected from a series of natural numbers $[n-F_{l-1}, F_l]$. k may be selected arbitrarily from this series.

As in the convention adopted in the aforementioned Gashkov et al, application, if k is selected as the left extremity of a series (least

allowable value), the a circuit is referred to as a left-side circuit; if k is selected as the right extremity of this series (greatest allowable value), the circuit is called right-side circuit. Also as described in the Gashkov et al. application, it is possible to distribute negations and execute other technological mapping techniques to derive 2-input elements in place of 3-input elements forming circuits that are topologically the same based on the following identities

$$h\_\{k+1\} = \neg(\neg h'\_1) \vee v'\_1 \& h\_k$$
$$= NAND(\neg h'\_1, ND(v'\_1, h\_k)),$$
$$\neg h\_\{k+1\} = \neg(h'\_1 \vee v'\_1, h\_k)$$
$$= AO6(v'\_1, h\_k, h'\_1), \text{ and}$$
$$V\_\{k+1\} = v\_\{1,k\} \& v\_k, \text{ where } A06(a,b,c) = NOR(AND(a,b),c).$$

In addition to using elements to form OR(NOT(a),b) in place of XNOR(a,b) elements, it is preferred to employ NOR and NAND elements instead of monotone OR and AND elements to reduce circuit depth and delay.

3. OPTIMIZATION TECHNIQUES

3.1. Distribution of Negations

For brevity symbol $f^{\wedge}\sigma$ designates the function f or its negation depending on whether σ = 1 or 0. A circuit with two outputs that realizes functions $h^{\wedge}\alpha\_n$ and $v^{\wedge}\beta\_n$ may be defined as $h^{\wedge}\alpha v^{\wedge}\beta$, where α=0,1 and β=0,1.

GP[α_1][α_2][β_1][β_2][β_3][β_4] designates a module with two outputs h and v and four inputs a, b, c, d, realizing functions AO21^{α_1}(b^{β_2}, c^{β_3}, a^{β_1}), AND^{α_2}(b^{β_2}, and d^{β_4}), where AO21(a,b,c) = OR(AND(a,b),c) (FIGS. 4 and 7B). For each of 64 modules, the optimal implementation is selected based on area or delay. As another example, module GP[0][0][1][1][1][1] may consist of elements AO6 and NAND, where AO6(a,b,c) = NOR(AND(a,b),c). See FIG. 7B. FIG. 8 illustrates a connecting module GP.

A module H^{α_1}V^{α_2}_n may be defined to provide functions h^{α_1}_n and v^{α_2}_n, where α_1 = 0,1 and α_2 = 0,1, and functions h_1,v_1,...,h_{n-1},v{n-1}. This module is generated recursively from modules H^{β_1}V^{β_2}_{k}, and H^{β_3}V^{β_4}_{n-k}, and their connecting module GP[α_1][α_2][β_1][β_2][β_3][β_4]. Similarly, a module H^α_n as may be defined to provide function h^α_n, where α = 0,1, and functions h_1,...,h_{n-1}. This module is recursively generated from modules H^{β_1}V^{β_2}_k and H^{β_3}_{n-k} and their connecting module AO[α][β_1][β_2][β_3].

The optimal distribution of negations is carried out as described in the aforementioned Gashkov et al. application.

3.2. Minimization of adder depth

Adders are frequently used as the internal modules in the more complicated circuits, such as multipliers. To calculate the depth of outputs of the adder located as an internal module in a larger circuit, it is not practical to assume that the depth of the adder inputs is equal to zero.

The depths of inputs A[O],B[0],...,A[n-1],B[n-1] of the adder are defined as a_0,b_0,...,a_{n-1}, b_{n-1}. The depth of the element providing functions U[i] = A[i]&B[i] and V[i] = A[i]∨B[i], for $0 \leq i \leq n-1$, is equal to U_i = max{a_i, b_i}+1, for $0 \leq i \leq n-i$. Therefore, the problem of minimizing the depth of an adder is reduced to one of minimizing the depth of circuit HV_n, realizing all functions h_1,v_1,...,h_n,v_n.

Minimization of the depth of circuit HV_n is carried out, by considering U[i] = 1, for $0 \leq i \leq n-i$ with the help of the optimal choice of parameter k in each step of recursive construction of HV_n from modules HV_{k}, HV_{n-k} and their connecting modules. The connecting modules consist of the parallel modules GP which are assumed, for example, to be formed from 2-input elements.

DH_n is the maximum depth of all the outputs of module HV_n, and Dh_n is the depth of the output of module HV_n that realizes function h_n. A dynamic programming algorithm is applied to minimize the

depth, based on the recursive application of the formulas

$$DH\_n = \min\{\max\{DH\_\{n-k\}+1, Dh\_k+2, DH\_k\}\},$$

$$Dh\_n = \min\{\max\{Dh\_\{n-k\}+1, Dh\_k+2\}\},$$

where DH_k and Dh_k are calculated for vectors of depths U_i, for $0 \leq i \leq k-1$, and DH_{n-k} and Dh_{n-k} are calculated for vectors of depths U_i, for $k \leq i \leq n-1$. If the minimum value of both DH_n and Dh_n are achieved for the same value of parameter k, that value is selected. Otherwise (if the minimum value of both DH_n and Dh_n are achieved for different values of parameter k), it is possible to accelerate the algorithm (at the expense of a possible diminution of exactitude) by always selecting the smallest value of k.

If exactitude is necessary, a vector DHh_n = (DH_n,Dh_n) is calculated during each step of recursion on the earlier calculated vectors DHh_{n-k},DHh_k. The calculation employs the formulas DH_n = max{DH_{n-k}+1,Dh_k+2,DH_k} and Dh_n = max{Dh_{n-k}+1,Dh_k+2} for the components of vector DHh_n. The values k are sorted to derive a set of minimum vectors DHh_n (concerning the relation of component-wise comparison). In further steps of the recursion, a search is conducted for a set of minimum vectors. The search examines all values of parameter k as well as all minimum vectors DHh_{n-k},DHh_k obtained on the previous step of recursion.

Vector DHh_k is recursively calculated for each k-dimension subvector (u_a,..., u_{b+k-1}) to calculate sequences of vectors DHh_n for a specific vector (u_0,..., u_{n-1}). In the simplest implementation of this algorithm the memory is equal to $0(n^2)$, and the complexity is equal to $0(n^3)$.

3.3. Minimization of Adder Fanout Depth

If the depth of a circuit is defined as the length of its maximum chain of elements in which the adjacent elements are connected to one another, the fanout depth can be defined as the maximum sum of fanouts of elements of such circuits, where the fanout of an element E is the number of elements having inputs connected to an output of element E. The delay of element E depends on the total capacitance of the indicated elements connected to it. The delay of circuit with given depth will be less if the fanout depth is less. Therefore to minimize the delay of a circuit, the fanout depth should also be minimized.

Smaller fanout depth can be achieved by changing the adder implementation approach (described in the previous section). The following describes implementation of an adder with smaller fanout depth, without changing the depth and only minimal increase in circuit area.

Consider a system of functions

HV^k_n = {h^1_k,v^1_k,...,h^{n-k+1}_k,
v^{n-k+1}_k},

where

$$h^i_k = h_k(U[i-1],U[i],V[i],...,U[i+k-2],V[i+k-2]),v^i_k = V[i-1]\&V[i]\&...\&V[i+k-2].$$

It is possible to recursively construct circuit HV^k_n from modules HV^m_{n-(k-m)} and HV^{k-m}_{n-m} and their connecting module, consisting of parallel located modules GP, based on the expansions

$$H^i_k = h_k(U[i-1],U[i],V[i],...,U[i+k-2],V[i+k-2]) = h^{i+m}_{k-m} \vee v^{i+m}_{k-m} \& h^i_m,$$

$$v^i_m = V[i-1]\&V[i]\&...\&V[i+k-2] = v^{i+m}_{k^m} \& v^i_m.$$

The resulting circuit is illustrated in FIG. 9. If k = F_l, the number m in the next step of a recursion can be selected as F_{l-1}, where {F_l} is the Fibonacci series. Then using mathematical inductions, the depth of circuit HV^k_n will be equal to l-1, fanout-depth will be equal to 3l-5, and the complexity of this circuit (the number of elements) is equal to 3n(l-2).

Fanout of each element realizing the function v^i{F_j} is equal to 3, and fanout of elements realizing the functions h^i_{F_j} is equal to 2. Each step of the recursion is performed using the same algorithm described in the aforementioned Gashkov et al. application to minimize parameter m. The circuit H_n is constructed recursively from modules H_K and HV^k_n and their connecting module,

consisting of parallel located modules GP, founding on the expansion

h_i = (U[0],U[1],V[1],...,U[i-1],V[i-1])

= h^{i-k+1}_kvv^{i-k+1}_k&h_{i-k}, i > k.

The depth of this circuit can be calculated as

DH_n = max{DHV^k_n + 1, DH_{n-k}+2},

and fanout-depth can be calculated based on a similar formula. The complexity of this circuit can be calculated from L(n) = 3n(1-2)+L(k), where

k = F_1 < n ≤ F_{1+1}, for F_1 Fibonacci numbers. Therefore, the asymptotic inequality

L(n) < 3(2+φ)n log_φ (n),

is obtained, where $\varphi = (\sqrt{5}+1)/2$ is an irrational number.

The adder based circuit is constructed using 2-input elements mapped from 3-input elements of a standard cell library. In the event 4-input elements are needed (as described in the aforementioned Gashkov et al. application), they may be constructed from 2-input elements.

The present invention thus provides an adder based circuit embodied in an integrated circuit. The adder includes an input module, a carry module and an output module. The carry module has a minimum depth defined by a recursive expansion of at least one function associated with the carry module based on a variable k derived from a Fibonacci series. Through use of invertor elements, XOR elements, XNOR elements and multiplexers selectively coupled to the input and

output modules, the adder based circuit can be configured to function as a subtractor, adder-subtractor, incrementor, decrementor, incrementer-decrementor or absolute value calculator.

The process of designing the adder based circuit includes defining at least one carry function of the carry module of the adder based circuit in terms of a Fibonacci series, and recursively expanding the carry function to find a minimum parameter of the Fibonacci series. Optimization of depth and delay of the adder based circuit are achieved.

In one form, the invention is carried out though use of a computer programmed to carry out the process. A computer readable program code is embedded in a computer readable storage medium, such as a disk drive, and contains instructions that cause the computer to carry out the steps of the process of designing a Fibonacci circuit in the form of a adder based circuit, including adders, subtractors, adder-subtractors, incrementors, decrementors, incrementor-decrementors and absolute number calculators.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.